REMARKS

In the foregoing amendments, claims 9 and 15 are amended. Claims 1-20 remain pending in the present application.

Response to the Use of Trademarks in the Specification

The Examiner has noted that several trademarks are used throughout the specification. In addition to those noted by the Examiner, others appear to be present. Each occurrence of all trademarks has been either replaced by generic terminology or capitalized with appropriate trademark notification markings in order to respect the proprietary nature of the marks. It is believed that all trademarks have now been properly identified in the specification. With respect to claims 9 and 15, the trademark TESTEXECUTIVE™ has been removed and replaced with generic terminology.

Response to 35 U.S.C. §102 Rejection

Claims 1-20 stand rejected under 35 U.S.C. §102(e) as allegedly being anticipated by *Grey et al.* (U.S. Patent No. 6,401,220). Applicants respectfully traverse this rejection on the grounds that *Grey et al.* does not disclose each element of the claimed invention.

Independent claim 1 is directed to an active data type for use in a computer program. The active data type comprises ***"at least a first algorithm associated with the active data type."*** Although *Grey et al.* discloses a mechanism that associates algorithms with step types, these algorithms are not associated with active data types. *Grey et al.* teaches an algorithm that includes code modules, pre-steps, and post-steps specifically associated with the step type, but fails to show any association of the algorithm to an active data type or a particular variable or value of the active data type.

More particularly, claim 1 further includes that the first algorithm is ***"automatically executed when an attempt is made to access a value associated with the active data type."*** In this regard, when a value of the active data type is accessed, either by a read or write operation, then the algorithm is executed automatically to process the value according to the algorithm. Although *Grey et al.* executes an algorithm, this algorithm is for the execution of routines or processes associated with

the step type. *Grey's* algorithms do not execute in response to a call for a specific value of an active data type, as claimed.

Independent claim 10 includes a computer program utilizing at least one active data type, wherein the active data type has ***"at least a first algorithm associated therewith."*** *Grey et al.* again does not disclose an algorithm associated with an active data type, but instead associates the algorithms with step types for executing pre-steps, test module, edit sub-steps, and post-steps. These associations are to functional processes, not values. Furthermore, *Grey et al.* fails to teach or suggest that the first algorithm is ***"executed automatically when an attempt is made to access a value associated with the active data type instance."*** Although *Grey's* algorithms may execute at a call for the step type, these algorithms are not called when an attempt is made to access a value associated with an active data type instance.

Claim 16 is an independent method claim including ***"automatically executing a first algorithm when an attempt is made to access a value associated with the active data type instance."*** Again, *Grey et al.* fails to disclose the execution of an algorithm when a variable or value associated with an active data type is accessed.

Grey et al. describes step types but does not describe a mechanism to process values of the step type on access of those values. It does not describe a mechanism that is capable of accessing the value of a specific data type during read requests or performing operations during write access for the value. Rather, it describes executable patterns, modules, or routines for a step type that associated with functional steps, such as pre-steps, post-steps, etc.

Grey et al. describes step types as providing configurable instances of routines, where each step type is relatively tightly coupled to the test routine or test module being called. In contrast, the active data types of the present application are not coupled with the routine being called. Variables of the active data type are associated with parameters in a way that does not require the routine to be changed to customize its behavior. Further, active data types can be used with numerous routines without requiring new type definitions, as is the case with step types.

The active data types of the present application have one or more value processing algorithms associated therewith for getting (read) or setting (write) their values. *Grey et al.* does not, in any way, refer to such value processing algorithms activated on access of an instance of such an active data type.

A proper rejection of a claim under 35 U.S.C. §102 requires that a single prior art reference disclose each element of the claim. *See, e.g., W.L. Gore & Assoc., Inc. v. Garlock, Inc.*, 721 F.2d 1540, 220 USPQ 303, 313 (Fed. Cir. 1983). In method claims, anticipation requires identity of the claimed process and a process of the prior art. The claimed process, including each step thereof, must have been described or embodied, either expressly or inherently, in a single reference. *See, e.g., Glaverbel S.A. v. Northlake Mkt'g & Supp., Inc.*, 45 F.3d 1550, 33 USPQ 2d 1496 (Fed. Cir. 1995). Since *Grey et al.* does not disclose every element or step of independent claims 1, 10, and 16, a proper 35 U.S.C. §102 rejection cannot be made. Therefore, Applicants respectfully submit that all claims are allowable over *Grey et al.* and request that the Examiner withdraw the rejection and pass this application to issue.

Dependent claims 2-9, 11-15, and 17-20 are believed to be allowable for at least the reason that these claims depend from allowable independent claim 1, 10, and 16. *In re Fine*, 837 F.2d 1071, 5 U.S.P.Q.2d 1596, 1600 (Fed. Cir. 1988).

CONCLUSION

In light of the foregoing amendments and for at least the reasons set forth above, Applicants respectfully submit that all objections and/or rejections have been traversed and/or accommodated, and that claims 1-20 are in condition for allowance. Favorable reconsideration and allowance of the present application and all pending claims are hereby courteously requested. If, in the opinion of the Examiner, a telephonic conference would expedite the examination of this matter, the Examiner is invited to call the undersigned agent at (770) 933-9500.

Respectfully submitted,

Glenn W. Brown

Glenn W. Brown
Reg. No. 51,310

THOMAS, KAYDEN,
HORSTEMEYER & RISLEY, L.L.P.
Suite 1750
100 Galleria Parkway N.W.
Atlanta, Georgia 30339
(770) 933-9500

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postage prepaid, in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on

Evelyn Sanders

Signature -

Attachments: Clean Copy of Substitute Specification
Marked-up Copy of Substitute Specification

* * * * * MARKED-UP COPY OF SUBSTITUTE SPECIFICATION * * * * *

AN ACTIVE DATA TYPE VARIABLE FOR USE IN SOFTWARE ROUTINES THAT FACILITATES CUSTOMIZATION OF SOFTWARE ROUTINES AND EFFICIENT TRIGGERING OF VARIABLE PROCESSING

TECHNICAL FIELD OF THE INVENTION

The present invention relates to an active data type and, more particularly, to an active data type used as a variable in a computer program that enables software routines to be augmented by providing a systematic technique for adding algorithmic processing to the data passed to or retrieved from those routines. In general, the present invention enables stand-alone, active data type variables to be defined as having values that are algorithmically calculated or determined when those variables are read and/or set. The active data type variable enables software routines to be easily customized and for the customized routines to be reusable. Furthermore, processing of the active data type variable is efficiently performed because processing preferably is only triggered when the values associated with the variable are read and/or set.

BACKGROUND OF THE INVENTION

The desire to make use of existing or new software routines for new applications is not new. Nearly every programming language, including C++, ~~Visual Basic~~VISUAL-BASIC™, ~~HP-Vee~~AGILENT-VEE™, and ~~LabView~~LABVIEW™ provide capabilities to invoke such routines. The routines themselves may be cast in a dynamically linked library form, an ~~Active-X~~ACTIVE-X™ component form, a ~~Corba~~CORBA™ component form, or a variety of others. The languages can be used to customize or fit the routines to be reused as needed. The customization cannot be extended across reuse instances of the customized routines. Furthermore, these solutions all require a considerable amount of programming skill to use effectively, and frequently require that a "wrapper" routine be written in the programming language in order to cast the original routine in a new role as a new routine ready to reuse.

#5

~~Test Executive~~Computer programs used to verify the integrity of electrical circuits, such as, for example, TEST-EXECUTIVE™ from DIT-MCO International Corporation, ~~TestStand~~TEST-STAND™ from National Instruments Corporation, and ~~TestExec SL~~TEST-EXEC™ from ~~Agilent Technologies~~Hewlett-Packard Company, are computer programs designed to enable users to reuse test and measurement software routines to create test programs and manage the testing operation. ~~Test Executives~~Computer programs used to verify the integrity of electrical circuits frequently provide language-like features that enable customization of the operation of those routines. However, like general purpose programming languages, they do not provide ways to systematically customize the operation of those routines across a number of reuse instances. To do this requires the same style of "wrapper" routines mentioned above, which are usually performed using a general-purpose language, such as those mentioned above, for example. This type of customization typically must be performed outside of the ~~Test Executive~~computer program development environment, and hence can be cumbersome.

As is generally known in computer programming, sometimes processing must be performed whenever the value of a variable changes. In most applications, this is accomplished by periodically checking for a new value in that variable and performing whatever processing is required when a new value is detected. This technique can be quite inefficient, especially in cases where the value changes infrequently and processing must be done quickly after a change is detected. In such cases, the value must be checked quite frequently and most checks correspond to wasted processing. Being able to trigger the processing at the point of change in the value of that variable would be more efficient, but doing so is not easily accomplished using existing solutions.

Accordingly, a need exists for an active data type that functions as a variable and that enables software routines to be easily customized. Furthermore, a need also exists for such an active data type that causes processing of the variable to be triggered in an efficient manner.

SUMMARY OF THE INVENTION

The present invention provides an active data type for use in a computer program. The active data type has an identifier and at least one algorithm associated therewith. The identifier is utilized by the computer program to identify an instance

of the active data type. The algorithm is configured to be automatically executed when an attempt is made to access a value associated with an instance of the active data type. The active data type may be a real, an integer, or a string, for example.

When the attempt to access the value associated with the active data type is made by a particular routine, the algorithm is automatically executed, which preferably determines the current value associated with the active data type instance. Preferably, the active data type has an identifier, a first algorithm and a second algorithm associated therewith. The first algorithm preferably automatically determines the current value of the instance of the active data type when a routine that utilizes the value of the active data type instance attempts to access the value. When the value of the instance of the active data type is set, the second algorithm preferably automatically post-processes the value to which the active data type instance has been set.

Preferably, a locking/unlocking mechanism is provided that locks the value of the active data type instance prior to invoking the particular routine and unlocks the value of the active data type instance once the routine has returned in order to allow a new value of the active data type instance to be set. By locking the value of the active data type instance when the routine is invoked and by unlocking the value when the routine returns, intermediate values generated by the routine are not set. Only the final value generated by the routine is set once the routine returns.

The present invention also relates to an apparatus for executing a computer program utilizing the active data type and to a method for utilizing the active data type in a computer program. These and other aspects and features of the present invention will become apparent from the following description, drawings and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram demonstrating the manner in which the present invention may be implemented in a ~~Test Executive~~test environment where test routines are utilized in conjunction with a ~~Test Executive~~computer program to test various types of instruments.

Fig. 2 is a block diagram demonstrating the manner in which the active data type of the present invention may be used in conjunction with format strings.

Fig. 3 is a block diagram demonstrating the manner in which the active data type of the present invention may be used in conjunction with a string format wherein

the values of symbols and parameters used in the string format may be set based upon a new string value extracted from a string format.

Fig. 4 is a state diagram demonstrating string formatting locking in accordance with one embodiment of the present invention.

Fig. 5A is a block diagram demonstrating the manner in which an arbitrary algorithm associated with a symbol may be executed to determine the value of a symbol whenever that value is requested by some other routine.

Fig. 5B is a block diagram demonstrating the manner in which an arbitrary algorithm associated with a symbol automatically processes the value of the symbol when some other routine attempts to set the value of the symbol.

DETAILED DESCRIPTION OF THE INVENTION

As will be seen from the following discussion, the active data type of the present invention enables the behavior of existing software routines to be customized in such a manner that the customized routine itself is capable of being reused, and the customization is extendable across instances of the customized routine. The active data type enables variable values to be algorithmically defined such that users of a particular active data type variable need not be aware of or concerned about the algorithm being invoked. This algorithmic definition is generated with the latest possible binding of the variable value, in that the value is calculated at the point of access. The present invention also enables algorithmic post-processing of the value of a variable to be triggered by variable value changes. This represents the earliest possible binding for post-processing the value as it is set. In addition, the present invention enables a parameterized general algorithm to be associated with the active data type for algorithmic processing of the active data type. All of these features of the present invention provide a technique for controlling how often the algorithmic processing associated with an active data type variable is performed.

The active data type variable of the present invention is useful for implementation with various types of the aforementioned ~~Test Executive~~computer programs used to verify the integrity of electrical circuits. Such ~~Test Executive~~computer programs are typically used by customers who have a body of existing test and measurement routines that they wish to make use of within the ~~Test Executive~~test environment. Other data-intensive environments, such as databases and spreadsheets, for example, may be other good application areas for the present

invention. Those skilled in the art will understand, in light of this disclosure, that the present invention is not limited to these practical applications. Those skilled in the art will understand that the present invention will have significant practical uses in virtually an infinite number of computer-related environments, either currently known or developed in the future. However, for purposes of illustration, the present invention will be discussed with reference to its use in a ~~Test Executive~~test environment for testing instruments and obtaining test measurement data.

Fig. 1 demonstrates a ~~Test Executive~~test environment in which the active data type variable of the present invention may be utilized with test routines in order to test instruments. A computer 10 executes a ~~Test Executive~~computer program 11 for verifying the integrity of electrical circuits. The ~~Test Executive~~computer program 11 performs various test routines 13 that are used to control various types of instruments, which are collectively referred to by block 14. The computer 10 is in communication with the data store 15 to enable the computer 10 to retrieve data from and store data in data store 15 associated with the execution of the ~~Test Executive~~computer program 11 and the various test routines 13. Those skilled in the art will understand that the data store might be a database, a collection of files, or other forms of data repositories. The manner in which the test environment illustrated in Fig. 1 can be implemented is known to those skilled in the art. The following discussion describes the active data type variable of the present invention and the manner in which it can be utilized in the ~~Test Executive~~test environment shown in Fig. 1.

As stated above, in accordance with the present invention, the test routines 13 can be customized utilizing the active data type techniques of the present invention and the customized test routines, and instances thereof, can be reused. The test routines 13 preferably are stored in the data store 15 and ~~envoked~~invoked by the ~~Test Executive~~computer program 11 when the computer 10 receives instructions from a user via the console of the computer 10. Therefore, the active datatype variables themselves and the test routines in which they are utilized are stored on some type of computer-readable storage device. Of course, the present invention is not limited with respect to the type of computer-readable storage device that is utilized for this purpose. Those skilled in the art will understand that a suitable storage device for such a purpose may be, for example, a magnetic storage medium, an optical storage medium, a solid state storage medium, or any other type of storage medium known now or developed in the future. The computer 10 comprises hardware or hardware

and software for performing the associated tasks. These tasks include execution of the ~~test executive~~computer program 11, execution of one or more test routines 13, customization of one or more of the test routines 13 in accordance with user input via the console of the computer 10, accessing the data store 15 and sending/receiving messages to and from the instruments 14 being tested.

One specific useful application of the active data type variable is its use in supporting string formatting (*i.e.*, both string value setting and string value parsing), as well as its use in arithmetic expressions to identify real numbers and integers. Other practical applications of the active data type variable relate to its ability to be configured to algorithmically select the correct calibration values to be used in a test measurement software routine based on parameters such as, for example, temperature, time, etc. For example, assuming processing of the active data type variable is triggered upon reading the value of the variable, the temperature parameter value being used in the routine may be algorithmically processed to determine or select calibration values for the routine.

Generally, the use of the active data types variables of the present invention would provide a generic method that would enable users to control message-based instruments without resorting to traditional programming. By using the active data type variable, users will be able to create routines (*e.g.*, measurement routines) that are functionally analogous to routines developed using a general purpose programming language. Moreover, the user will be able to easily generate the measurement routines themselves without having to utilize a general purpose programming language. As stated above, generating such routines by utilizing a general purpose programming language typically requires that the user write a wrapper routine, which can be tedious. Furthermore, the user would be required to know the particular programming language in order to write the wrapper routine.

The active data type variable of the present invention provides several desirable features. By causing processing of the active data type variable to occur when the variable is read, or accessed, the binding of algorithmically-defined variable values can be postponed until they are needed. This, in turn, enables the frequency with which the algorithmic processing is performed to be controlled to avoid unnecessary or undesired processing. Likewise, by causing post-processing of variable values to occur when the values change, the frequency with which the post-

processing is performed can be controlled to avoid unnecessary or undesired processing.

Utilizing the active data type variable (*i.e.*, a variable with parameters/ properties that are active) enables existing software routines to be easily customized without having to resort to traditional programming. These customized versions of the software routines can be reused like (and generally will look like) any other routine. Furthermore, universal extensibility of routines independent of the implementation or language used for that routine can also be realized. Behavior extensions preferably are stored and managed with the data associated with the routine without the knowledge or cooperation of the routine being extended. In addition, the triggering of variable processing in accordance with the preferred embodiment of the present invention result in performance advantages in routines that have variables that are seldom accessed.

A description of the construction and operation of the active data type of the present invention will now be provided. Active data types in accordance with the preferred embodiment of the present invention are comprised of the following components:

(1) Zero or more extended properties stored with each active data type instance in order to control the active data type algorithms;

(2) A data type editor that knows about the active nature of the data type and exposes the extended properties for user control;

(3) The algorithms associated with the data type instance "get" and "set" operations; and

(4) A general data type management infrastructure to invoke the data editor, save and restore data type instances with associated extended properties, prepare the execution environment for active data type algorithmic processing, control access to the data instance, and invoke the algorithms on data access.

It can be seen from this description of the active data type of the present invention that active data type variables can be defined based on an arithmetic expression involving other ~~Test Executive~~ user and system variables. These expressions preferably are invoked when the variable values are retrieved, and hence can provide the most up-to-date value for the expression. In contrast, expressions in general purpose programming languages are typically invoked once, namely, when

the value of the variable is set. If the values of the variables in those expressions change, the changes are not reflected until the expression is explicitly re-executed.

In addition, active data type string formatting can be performed in accordance with the present invention to enable arbitrary parameters within the ~~Test Executive~~test environment to be incorporated into the message (strings) sent to instruments to control them. In the ~~Test Executive~~circuit-integrity-verifying program developed by Agilent Technologies Company, known as TestExecSL, these messages are sent to instruments by generic message-send and message-receive routines supplied with TestExecSL program. In accordance with the present invention, string data types used in such an environment can be made active by providing formatting control. The formatting control feature is defined by the following fundamental extended properties:

(1) Format: a string specifying the format of the string data type instance (*e.g.*, a reference to "Variable 1" may be specified as "%Var1%"); and

(2) Format operation: the type of formatting operation associated with the string data type instance. For example, the format operation "Set String" may indicate the format to be used to calculate the value of the string. Similarly, the format operation "Scan String" may indicate the format to be used to extract subvalues from the value to which the string is set.

The underlying algorithms of the active data type variable preferably operate in a manner very similar to the manner in which the "sprintf" and the "sscanf" functions in the C programming language operate. However, the invocation of these functions is controlled in accordance with the aforementioned active data type components on access to the data instance rather than by explicit function calls. The "Set String" formatting operation occurs when any routine reads the value of the data type instance. The "Scan String" formatting operation occurs when any routines change the value of the data type instance. As stated above, the components of the active data type, such as the formatting algorithms and string data editor, enable the users of ~~Test Executive~~test program to control the value of the string using a set-string format or extract subvalues from the string using a scan-string format.

The following discussion of Fig. 2 provides details on active data type implementation in the ~~aforementioned TestExec SL~~circuit-integrity-verifying program. The ~~TestExec SL~~test program provides support for the use of named symbols and parameters of a variety of types, including string data type. For the

string data type, the ~~TestExecSL~~test program supports associating a string format with any instance of a string in a manufacturing test program. This string format can either algorithmically specify the value of the string or specify how to extract portions of a changed value of the string to set the values of other symbols or parameters defined in the manufacturing test program. A string format can specify how to construct the value for a string from other symbols or parameters. A string format in accordance with the present invention may conceptually look like the string format shown in Fig. 2. In this case, the parameter has a name "Parm1" and a value of 5. The % signs are used to delimit the parameter name in the string format. For simplicity, Fig. 2 does not illustrate the algorithm associated with "Parm1" for generating the value associated therewith.

Fig. 3 demonstrates the manner in which the values of Symbols and Parameters may be set from a New String Value. A string format can specify how to extract values from portions of a new string value in order to set the values of other symbols or parameters. Therefore, when a message is returned from a test instrument, it can be parsed and symbols and/or parameters used elsewhere in the test program can be set in accordance with the values obtained during the parsing procedure. The new string format may then constitute a new message to be sent to the same or a different test instrument within the same or a different test routine, respectively. In the example shown in Fig. 3, the parameter "Parm1" is set equal to the value extracted from the new string value.

Another aspect of the present invention relates to locking and unlocking of string formatting during invocations of routines associated with the active data types. This aspect of the present invention will now be discussed with reference to the state diagram shown in Fig. 4. If a particular string parameter is accessed frequently, performing string formatting each time the string is accessed may be too slow for the particular routine being performed. It also may be disruptive for an existing routine to have the string value change dynamically while that routine runs. For both of these reasons, the ~~Test Executive~~test program of the present invention calculates the string value before executing a particular routine, then locks that value for the duration of that routine's execution (*i.e.,* for that instance of the routine), regardless of the number of times the string value is retrieved. For the case where new string values set the values of other symbols or parameters, string value changes occurring during the execution of an existing routine are ignored. After that routine completes,

~~TestExecSL~~the test program 'unlocks' the formatting, allowing the ending string value to determine the values of any symbols or parameters referenced by the format.

This behavior is demonstrated by the state diagram of Fig. 4. In accordance with this diagram, the state of the string data instance starts at the 'Unbound' state (not shown), progresses to the 'Bound' state 21 as its location in memory is fixed, proceeds to the 'Action Bound' state 23 when a routine (user routine) is associated with it, and, finally, proceeds to the 'Action Locked' state 25 when the string value is fixed based on the string format, if any. The reverse state progression occurs when the Action returns control to the ~~Test Executive~~test program (*e.g.*, TestExecSL). In transitioning to the 'Action Bound' state 23 from the 'Action Locked' state 25, the string value is used to calculate the values of other parameters and symbols if there is a string format and if the correct 'Scan String' format operation has been requested.

While the TestExecSL implementation of the present invention makes use of active data types for string formatting only, the active data type of the present invention has more general usability, as will be understood by those skilled in the art. An arbitrary algorithm could be executed to determine the value of a symbol whenever that value is requested by other software. Similarly, an arbitrary algorithm could be executed to make use of a new value whenever the value of a symbol is set. Fig. 5A demonstrates the former general implementation of the present invention whereas Fig. 5B demonstrates the later. As shown in Fig. 5A, when a user routine 31 attempts to obtain the value of symbol "A", an arbitrary value processing algorithm 33 associated with the symbol "A" 32 is automatically executed to determine the value of the symbol. The value of the symbol is then sent to the user routine 31. As shown in Fig. 5B, when the user routine 31 sets the value of the symbol "A", an arbitrary value processing algorithm 35 associated with the symbol "A" processes the value to make use of the new value. The active data type in this case is symbol "A" and has all of the aforementioned attributes of the active data type of the present invention.

The present invention has been described with reference to particular embodiments, namely, with respect to its implementation in a ~~Test Executive~~circuit-integrity-verifying program environment. However, as stated above, the present invention is not limited to this particular implementation. Those skilled in the art will understand that many variations may be made to the embodiments and

implementations mentioned above without deviating from the spirit and scope of the present invention.